A data pipeline for Optical and Near-infrared Solar Eruption Tracer

Xinhua Wang[1,2] , Dong Chen[1] , Tao Deng[3] , Hongbing Dai[3] , Yongyuan Xiang[1]

(1. Yunnan Astronomical Observatories, Chinese Academy of Sciences Kunming Yunnan 650011;
2. University of Chinese Academy of Sciences Beijing 100049;
3. School of Information, Yunnan University kunming Yunnan 650504)

Abstract: With the advent of large astronomical equipments, the traditional development model for data reduction faces problems such as redundancy of programs and conflicting environmental dependencies; Besides as a cluster is a highly coupled computing resource, serious environmental conflicts can lead to the unavailability of the entire cluster. To address this problem, we have developed a new pipeline framework using the concept of microservices. This paper presents the ONSET data pipeline developed through this framework. To achieve near real-time data processing, we optimize the core program using MPI and GPU technologies and evaluate the final performance. The results show that this development model can be built in a short time to meet the requirements of the pipeline, and we believe that this development model has implications for future multi-band and multi-terminal astronomical data processing.

Keywords: ONSET; data pipeline; container; GPU

Chinese library classification number: P182.2+1 TP751.2   Document code: A

1. Introduction

Large astronomical facilities lead astronomy to the era of big data. How to process these data become a major problem before the astronomical community at present. Hence, data pipeline is one of the key means for astronomers to understand the data.

Normally, a data pipeline will be developed in either monolith or modular way.The pros and cons of these 2 approaches has been discussed for years. And modulization design for data pipeline is the main stream up to now.Besides, deploying a data pipeline is rather a hard work for most of times, as the dependency and other problems may arise with regard to OS of your platform.

In order to tackle this situation, we try to apply a new framework based on virtualization technology for data pipeline development for ONSET at FSO. Some details of this new framework will be discussed in this paper. In section 2, we introduce the basic information of ONSET, some details of data pipeline based on new framework is presented in Section 3. Then in section 4 and 5, we show result of using CUDA for near-realtime data processing. Finally, we briefly discussed some possible improvements for the data pipeline for ONSET.

*Foundation item：Supported by the National Natural Science Foundation of China (U1831210,11973088)
First author: Xinhua Wang Email: wangxinhua@ynao.ac.cn

Corresponding author:Dong Chen Email: chd@ynao.ac.cn

2. Overview of ONSET

The Fuxian Lake Solar Observatory (FSO) is located in Yunnan, China. FSO has two telescope, Optical and Near-infrared Solar Eruption Tracer (ONSET)[3] and 1m New Vacuum Solar Telescope (NVST). ONSET is jointly developed by Yunnan Observatory and Nanjing Institute of Astronomical Optics Technology, and funded by Nanjing University. Its effective diameter is 275mm and observe the sun in three wavelength: He I 10830Å, Ha and white-light at 3600Å and 4250Å.show in Tab.1.The corona, chromosphere and photosphere can be observed simultaneously in a full-disk or partial-disk solar with a field of 10''. At present, there are two acquisition cameras, namely Andor Neo and Flash4.0 V3. Neo' s conventional observation mode is to collect 10 sets of full-plane or partial images of the sun every minute, with an image size of 852x852; Flash's regular observation mode is to collect 10 sets of full-plane or partial images of the sun every 30 seconds, with an image size of 1700x1700. 150 frames per group .data volume is shown in Tab.2.

Since the data of ONSET and NVST have similar data processing flow, the pipeline framework designed can be used to develop the pipeline of ONSET. This has the advantage of reducing repetitive work and accelerating pipeline deployment. Next we describe the development of the ONSET pipeline.

Table 1 Channels of ONSET

Channels	Wavelength(Å)
Ha	6562.8±5 Tunable
White light	3600，4250
Near ingrared	10830±0.85 Tunable

Table 2 Sampling rate of ONSET

Camera	Data(8h/day)	Data Volume(200days)
Andor Neo	65GB	12.7TB
Flash4.0	258GB	50.4TB

3. Data pipeline for ONSET

At present, data products of ONSET are classified into three levels:

- Level 0 data: raw data, unprocessed data collected by the cameras.
- Level 0.5 data: level 0 data calibrated by flat and dark processing, then using frame selection and reconstruction by speckle interferometry and speckle masking.
- Level 1 data: According to filed situation of observations, add headers to level 0.5 data and save them in FITS format. Level 1 data is science-ready data and can be used for scientific research.

Transfer Node is responsible for saving the raw data to the storage device and sharing the mounting point to computing center via a high-performance network (10GbE). Finally, the raw data is processed into science-ready data by computing center. Then science-ready data is archived through Distribution Node that show in Fig.1.

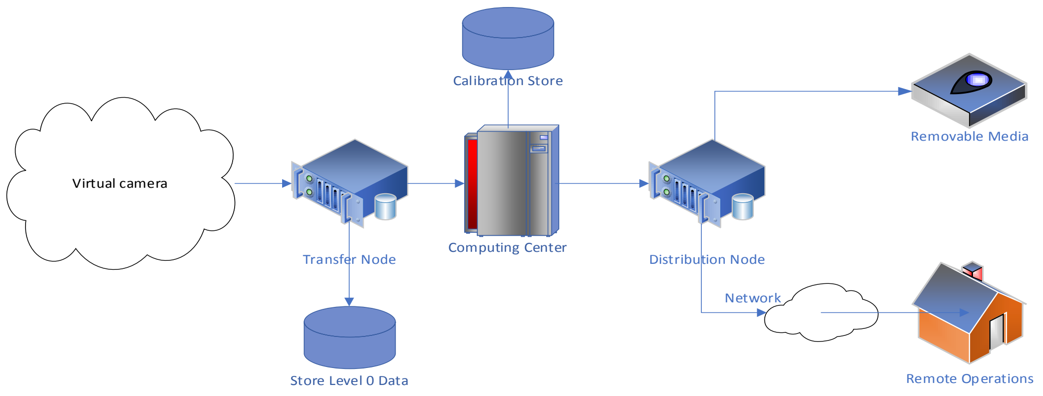


Fig.1. Diagram of ONSET hardware structure

In October 2020, ONSET decided to develop a new data pipeline. The requirements of this new pipeline is as follows:

1) Using speckle-masking method to process all the data acquired at ONSET;

2) At the beginning of development, ONSET will provide a Dell R730 server (56cores + K40G PU) for test; and finally the 8-hrs daily observation is required to be processed within 3-5 days after observation;

3) The development of data pipeline will be based on Python;

4) The quality of science ready data will be assessed by the scientist from ONSET;

3.1 Algorithm

At present, speckle mask[4] and speckle interferometry[6] are the most widely used in high-resolution solar image reconstruction, The implementation is as follows. Speckle interferometry is used to reconstruct amplitude and the speckle masking reconstructs the phase. The high-resolution reconstructed images with this algorithm were termed Level 0.5 data at ONSET, or science-ready data. The algorithm flow chart is shown in Fig.2.[7]

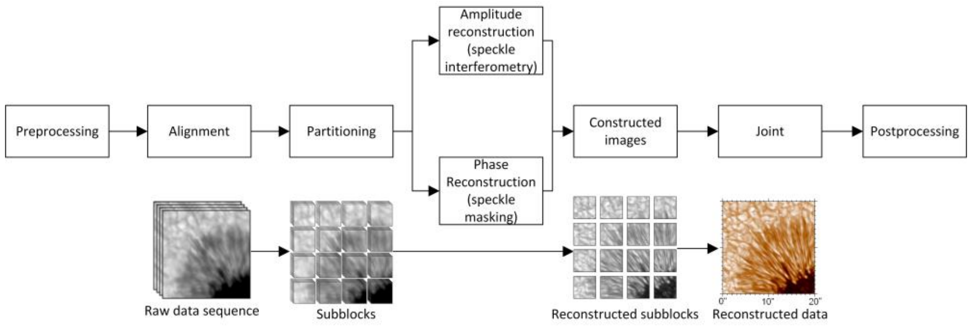


Fig.2. Level 0.5 algorithm of ONSET

3.2 Pipeline configuration file

The Pipeline configuration file provides the ability for a user-defined pipeline structure in YAML format. The YAML format is a highly readable format for expressing data serialization and is ideal for writing configuration files. The keywords of pipeline configuration file is shown in Tab.3.

The data exchange of microservices in data pipeline is based on ZeroMQ's message forwarding mechanism, so users need to determine the connection relationship between microservices and bind ports and required resources through pipeline configuration file then **Pipeline Parser** will generate executable script according to the pipeline configuration file, support scaling and automatically submit tasks.

Table 3 pipeline configuration file keywords

keywords	Value	Comment
Pipeline	"*appname"	microservices name
parameters	"nodes"	number of nodes required
	"ppn"	number of processors on a single node
	"mem"	required memory
	"scheduler"	resource scheduler(PBS/SLURM)
	"workdir"	working directory
	"savedir"	saving directory
	"sitf"	path of speckle imaging transfer function
	"script"	path of script that created by PipePaser
*app	"name"	name of apps
	"volume"	mount the host disks to singularity
	"sif"	path of singularity image
	"port"	mapped port
	"retrieval"	find interested files implemented by glob modular in python
	"branch"	core functions
	"archive"	path of processed data
	"request"	requested resources
	"mpi"	use mpi or not
retrieval	"loc"	expression
	"notin"	filter

3.3 Containerization of microservices

The most important concept in our new framework is microservices, which first came from the Internet and refers to applications that can handle specific requests and are relatively independent of each other. One of the benefits of microservices-based development is to decouple complex functional relationships into multiple subfunctions that are single-functional and easy to maintain, so that the number of corresponding microservices can be dynamically increased or decreased to achieve scaling when the load changes. More and more programs moved from the original host environment or virtual machine environment as a carrier to the container as a carrier[1][2]. After research we choose singularity, a high-performance container technology developed by Lawrence Berkeley National Laboratory specifically for large-scale, cross-node (High-Performance Computing)HPC and (Deep Learning)DL workloads[5].

Our framework is to combine the term of microservices and functions of pipelines wrapped with singularity. Each microservices is a function in the data reduction wrap in a singularity image. Besides other known advantages, another advantage of this framework is to increase the performance of the pipeline through scaling by using message passing model. In theory, GPU have more computing power than CPU but to achieve 100% GPU performance requires proficiency in grid computing and other professional GPU programming knowledge. In order to release the power of GPU, our solution is to use message preemption mechanism, which could increase GPU resource utilization through scaling by means of peripheral acceleration and significantly reduce data processing time. The data processing speed is linearly related to the number of microservices without exceeding the graphics memory, as shown in Fig.3. It can significantly improve GPU resource utilization when GPU resource utilization is low.

The carrier of microservices is containers. The introduction of container technology in the pipeline solves the environment dependency problem that has long plagued developers and further increases the portability. The data pipeline consists of two modules:

- Coordinator :prepare data for other microservices
- Pipeline Parser: Parse the defined pipeline configuration file, turn it into a PBS, Slurm or standalone script and submit the task

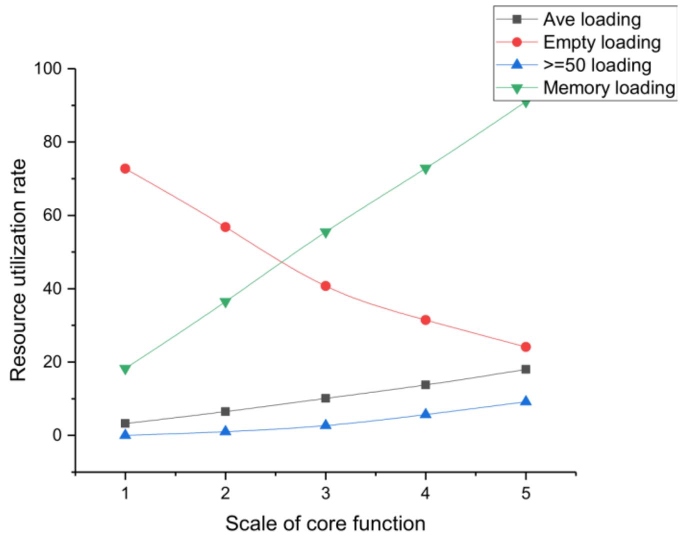


Fig.3. Utilization rate of the GPU

We decouple the pipeline into microservices as follows

- Dark: Dark fielding the raw data.
- Flat: Flat fielding the raw data.
- matching: matching the data with off band.
- core1: level1 algorithm for 6563 Å.
- core2: level1 algorithm for 3600 Å and 4250 Å

The pipeline structure diagram based on the container is shown in the Fig.4.

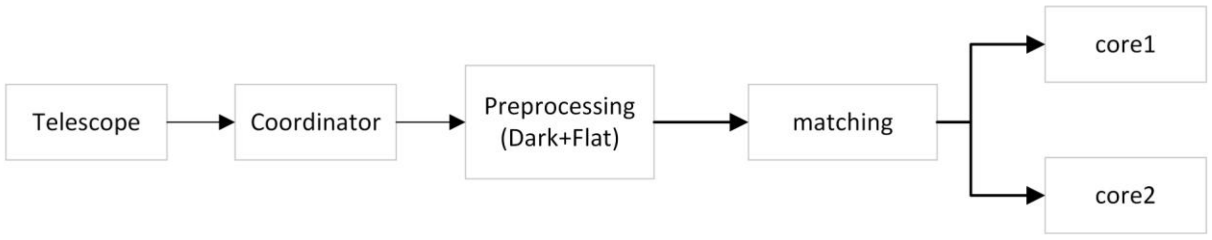


Fig.4. The pipeline structure diagram based on the container

microservices bind port through *Pipeline_API*, the main programs are shown in Tab.4.

Table 4 Code implementation of bind port through *Pipeline_API*

Code implementation of bind port through *Pipeline_API*

```
import Pipeline API as PA                                             #basic library for HPVDP
def AppProcess(pipefile):
    pipeline = PA.read pipe(pipefile)             #convert the yaml format to a dictionary format
    port in = pipeline[ ’Pipeline’ ][ ’* appname’][’port’ ][ ’in’ ]          #input port
    port out = pipeline[ ’Pipeline’ ][ ’* appname’][’port’ ][ ’out’]        #output port
    recv = PA.bindport(port in,’in’)                                       #bind the input port
    send = PA.bindport(port out,’out’)                                     #bind the output port
```

4 GPU Processing within mpi4py

It will take a large number of short-exposure images with high-frequency information during every observation at ONSET. Then we adopt the speckle masking method to reconstruct the phase of the image, and speckle interferometry to reconstruct the amplitude of the image. The speckle mask method involves the calculation of a 4-dimensional double spectrum and phase recursion, which are very time-consuming. Hence we tried to use GPU and MPI technology to speed up the processing time of the program. The flow chart of core algorithm is shown in Fig.5.

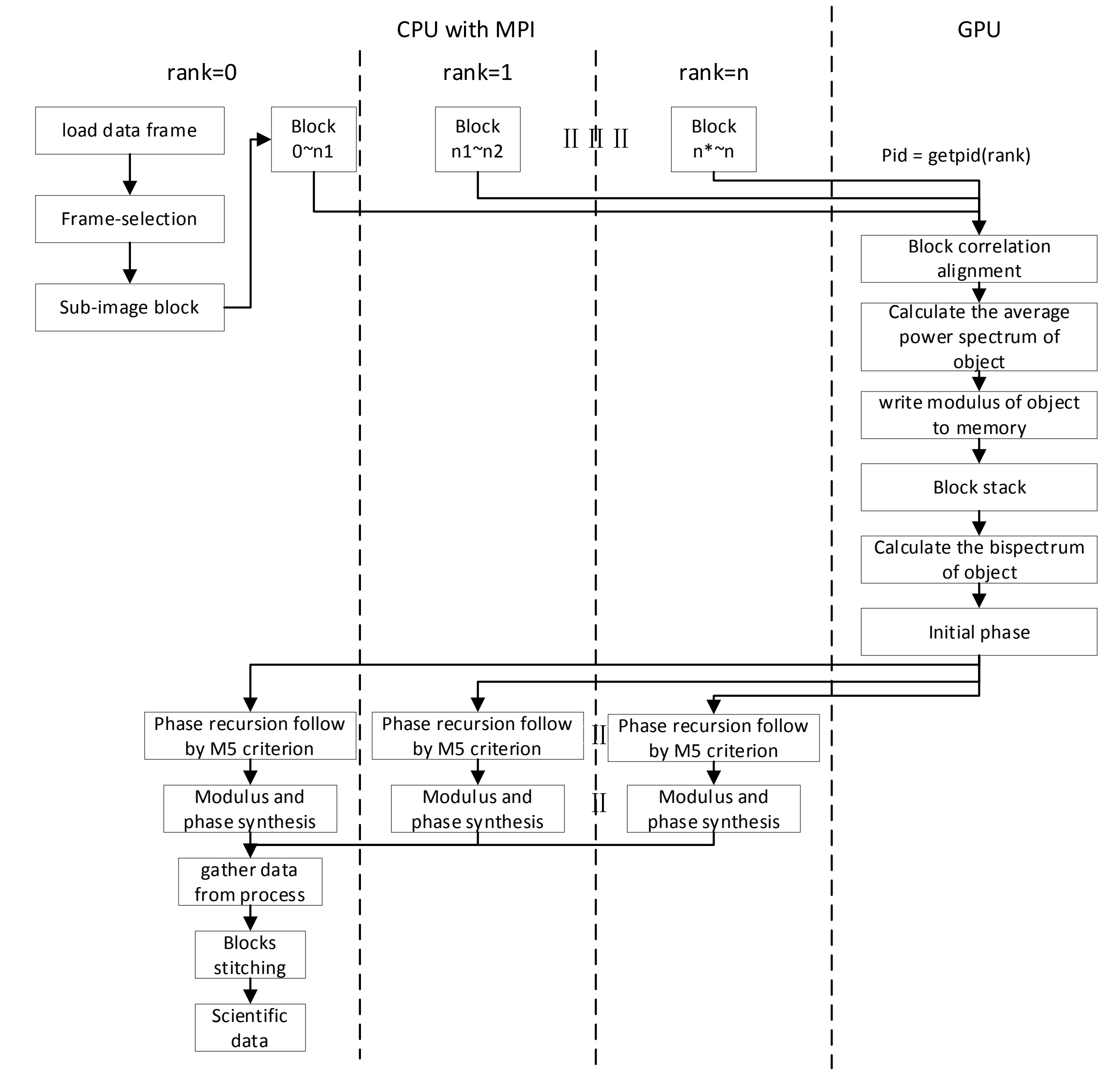


Fig.5 Flow chart of core algorithm based on MPI and GPU

Within the ONSET program, the whole flow contain these steps: image preprocessing, initial image alignment, seeing estimation, speckle interferometry transfer function calculation, image block processing, image stitching, and one of the most time-consuming process in the entire program is the image block processing. After analysis, we decide to use the CUDA architecture from Nvidia GPU to execute block processing in parallel. While improving GPU utilization, we also found that CPU utilization is at a low level. In order to improve CPU utilization, the MPI programming model is adopted. Fig.6. provides common methods of MPI.

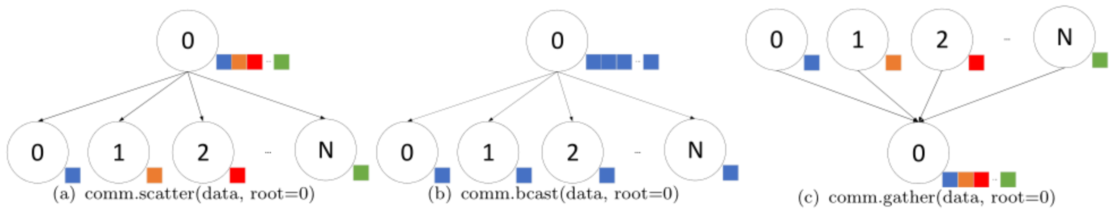

(a) comm.scatter(data, root=0)
(b) comm.bcast(data, root=0)
(c) comm.gather(data, root=0)

Fig.6. communication category of mpi4py

In MPI programing model, one important thing to do is to scatter tasks to each node. In our case, that means to scatter task of image sub-block processing all over the nodes. The master node will divides the aligned observation images into sub-block groups; then it will assign the sub-block calculation tasks to the slave nodes. The master and slave nodes are responsible for calculating the sub-block processing, and the result of the master and slave node calculation will return to master node. But there is a limitation in MPI which impedes parallelization of data chunks with more than 2^{31}=2147483648 elements because the MPI standard uses C *int* for element count, which has a maximum value of 2^{31} for positive value[8]. When object bigger than that, the function will report an error. To avoid this, we divide the whole data into two part for separate processing. To make a reasonable task allocation, it will first need to calculate the length of the task by the master node, and divide the length of the task by the number of cores involved in the calculation. If it can be divided, we use the scatter function and send it to each slave node evenly. If it can't be divided, this article uses the method of adding an empty array to make it can be divided, and then use the scatter function to evenly send it to each slave node. The subsequent calculations will start, and each node returns the corresponding result part to the master node.

Algorithm 1 Strategy for scatter

```
1:  procedure DIVIDE_ TASKS(tasks_list, cores)
2:      task1_list ← None
3:      task2_list ← None
4:      cores ← cores
5:      if comm_rank == 0 then
6:          task1_list ← tasks[: int($\frac{1}{2}$ * len(tasks_list))]
7:          task2_list ← tasks[int($\frac{1}{2}$ * len(tasks_list)) :]
8:          if len(task1_list)//cores != 0 then
9:              Padding(task1_list, 0, cores)
10:         else if len(task2_list)//cores != 0 then
11:             Padding(task2_list, 0, cores)
12:         end if
13:     end if
14:     thread_tasks = scatter(task1)
15:     result_mid1 = StartReconstructed(thread_tasks)
16:     result_final = gather(result)
17:     thread_tasks = scatter(task2)
18:     result_mid2 = StartReconstructed(thread_tasks)
19:     result_mid3 = gather(result)
20:     APPEND(result_final, result_mid3)
21: return result_final
22: end procedure
```

5. Result

In order to test our data pipeline, we used GPU+CPU and CPU to process the WH data for test. The whole environment is list here:

Hardware: Intel(R) Xeon(R) Gold 5115 CPU @ 2.40GHz x2, 128GB RAM, GeForce RTX 2080Ti 11GB x4;

Software: Ubuntu 16.04.6 LTS,CUDA 10.1, GPU driver 430.40, Singularity 3.5.0, OpenMPI 1.10.7, Python 3.6.10.

Ten sets of 50 frames (1700x1700pixel) of WH-band data on October 10, 2020 were selected, and each frame was chunked in an overlapping fashion with 2704 blocks (96x96pixel) in each set.The execution time is averaged for ten sets of data. Experiments have shown that CPU+GPU can improve program execution speed compared to CPU mode. results are shown in Tab.5. The introduction of MPI and GPU not only brings about performance improvement, but also consumes some time due to data preparation and MPI initialization. The statistical results are shown in Tab. 6. The result is that a small amount of data transfer between the CPU and the GPU does not incur significant performance costs but using MPI to collect data and initialize parameters imposes a significant performance overhead.

Table 5 Optimization results in two methods

Pattern	method	Elapse time(s)
Serial	CPU	4607
	CPU+GPU	3150
Parallel(MPI with 20 cores)	CPU	477
	CPU+GPU	360

Table 6 Elapse time in data preparation and initial mpi

Data preparation and initial mpi(CPU+GPU with 20 cores)		Elapse time(s)
Data transmission between host(CPU) and device(GPU) in the pipeline	Raw data(host to device)	1.02
	Data for bispectrum and Modulus(host to device)	0.1
	Data for phase recursion (device to host)	0.39
Initial MPI		9
Gather data by MPI		40

To test the efficiency of the program optimization, we tested the effect of the number of processes on execution efficiency in GPU+CPU mode and CPU mode, and the results showed that the program running efficiency is less sensitive to the number of processes than in CPU mode when using GPU+CPU mode.Fig.7 shows the result. In order to show the reconstruction results, we selected HA and White band data respectively. The result is shown in Fig.8

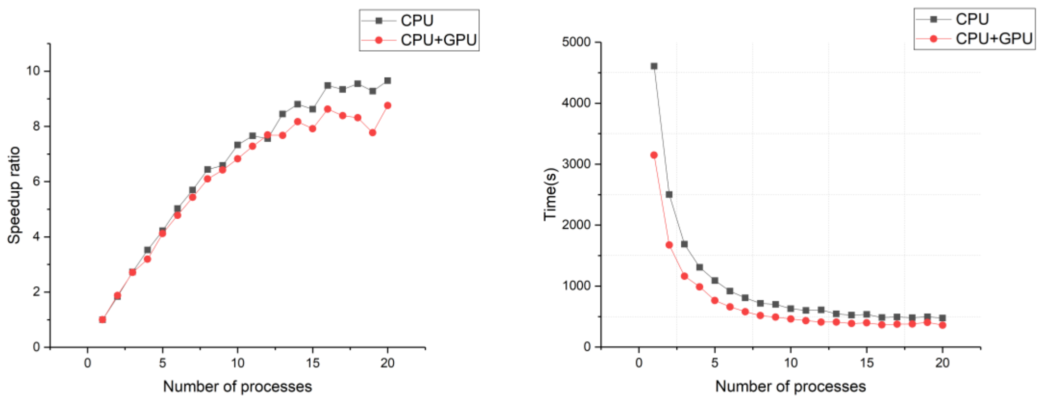
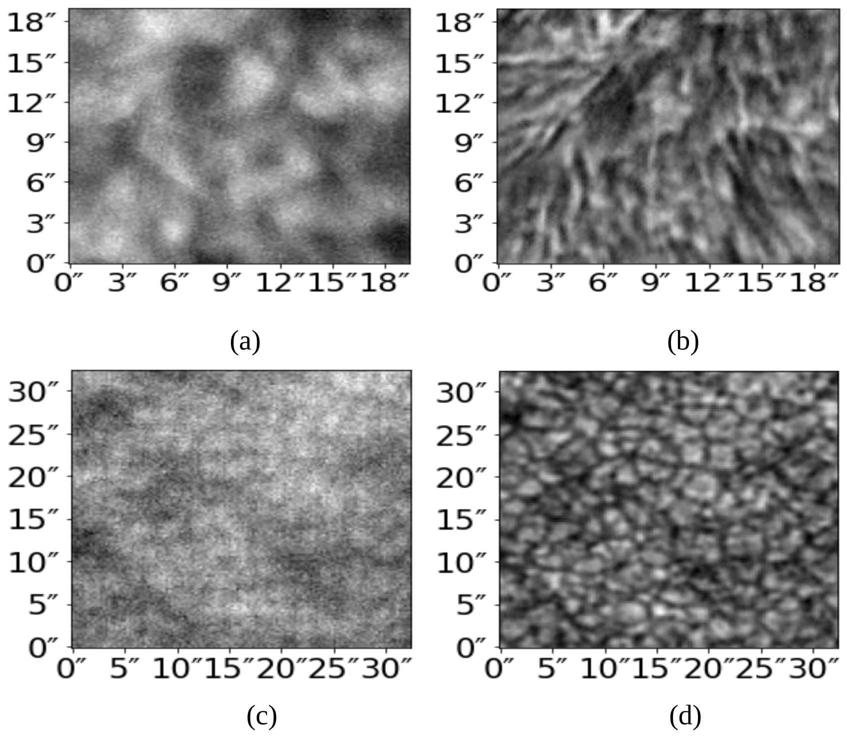


Fig.7. Acceleration ratio(left) and execution time(right) in CPU and CPU+GPU mode with different number of processes



Note: HA band on the top, white light band on the bottom

Fig.8. Original images(left) vs Reconstructed images(right)

6. Conclusion& Discussion

In this article, we developed the prototype data pipeline using a container-based pipeline framework for ONSET and optimized the original CPU program using the GPU, resulting in a significant speedup of the original program. We are planning to deploy the data pipeline in near future. ONSET's GPU server contains a tesla K40m GPU and Xeon(R) CPU E5-2680 v4 @2.4GHz(56 cores). It takes 53s to reconstruct a set of HA-band images and 250s to reconstruct a set of WH-band images in this environment, so a day's worth of data (8h) takes about 14h(HA) and 66h(WH) to process respectively. And the timing requirement for data reduction at ONSET is reached.

For the whole image reconstruction at ONSET, the multiple layers of judgement and loops required by the image phase reconstruction is the most time-consuming.Obviously this logical operation is more efficiently executed by the CPU than the GPU, we are working on how to speed up this process with CPU now. In general, further improvements to the algorithm are necessary to enable near-realtime processing.

From our practice, it was found that this development model allowed for flexible modifications to the microservices, which enables to add more required functionality to the pipeline in time without having to modify the entire program. We believe that this container-based development model will save a lot of time both in the development and deployment of the astronomical data pipelines, while astronomical facilities are now moving towards multi-terminal and multi-wavelength.

7. Acknowledgement

This work is supported by the National Natural Science Foundation of China (U1831210, 11973088).

References

[1]. Liu Ying, Zhang Mo, Huang Maohai. Astronomical Data Archive System Test Based on Docker Technology. Astronomical Research and Technology, 2020, 17(2): 217-223.

[2]. Yao Kun, Dai Wei, Yang Qiuping, Mei Ying, Shi Congming, Wang Feng. Automatic Deployment Method of Astronomical Application Software Based on Container Technology. Astronomical Research and Technology, 2019, 16(3): 321-328.

[3]. Cheng Fang, et.al. A new multi-wavelength solar telescope: Optical and Near-infrared Solar Eruption Tracer(ONSET).Research in Astron. Astrophys,2013,13(12):1509-1517

[4]. Lohmann, A.W., et.al. Speckle masking in astronomy: triple correlation theory and applications. Appl. Opt. 22,4028-4037

[5]. Kurtzer, G.M, et.al. Singularity: Scientific containers for mobility of computer. PLOS ONE 12, 1-20.

[6]. Weigelt,G., Modified astronomical speckle interferometry “speckle masking”. Optics Communications 21, 55-59

[7]. Xiang Yongyuan .et.al. Reconstruction method of high resolution solar image. Progress In Astronomy, 2016,34(1)

[8]. Alex M. Ascension and Marcos J. Araúzo-Bravo. BigMPI4py: Python module for parallelization of Big Data objects. bioRxiv, (2019).

ONSET 数据流水线

王新华 1,2，陈东 1，邓涛 3，代红兵 3，向永源 1

（1.中国科学院云南天文台 云南 昆明 650011；2.中国科学院大学 北京 100049；3.云南大

学信息学院 云南 昆明 650504）

摘要：随着天文大科学设备的投入,传统的开发模式面临程序重复开发，环境依赖冲突等问题；另外，集群是一个高度耦合的计算资源，严重的环境冲突可能导致整个集群的不可用。为了解决这个问题，我们采用微服务的概念开发了新的流水线框架，这种框架可以实现短期内开发和部署新的流水线。本文介绍了通过这种框架开发的 ONSET 数据流水线，为了实现近实时数据处理，我们采用 MPI 和 GPU 技术对核心程序做了优化，并对最后的性能做了评估。结果表明这种开发模式可以在短期内搭建满足需求的流水线，并且我们相信这种开发模式对于未来多波段多终端的天文数据处理有借鉴意义。

关键词：ONSET;数据流水线;容器;GPU